CS 188: Artificial Intelligence
Spring 2010

Lecture 6: Adversarial Search

2/4/2010

Pieter Abbeel – UC Berkeley

Many slides adapted from Dan Klein

Announcements

- Project 1 is due tonight
- Written 2 is going out tonight, due next Thursday

Today

- Finish up Search and CSPs
- Intermezzo on A* and heuristics
- Start on Adversarial Search

3

CSPs: our status

- So far:
  - CSPs are a special kind of search problem:
    - States defined by values of a fixed set of variables
    - Goal test defined by constraints on variable values
  - Backtracking = depth-first search with incremental constraint checks
  - Ordering: variable and value choice heuristics help significantly
  - Filtering: forward checking, arc consistency prevent assignments that guarantee later failure
- Today:
  - Structure: Disconnected and tree-structured CSPs are efficient
  - Iterative improvement: min-conflicts is usually effective in practice

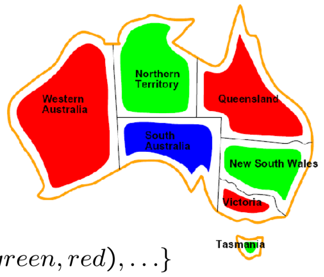
4

Example: Map-Coloring

- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$
- Domain: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

$$WA \neq NT$$

$$(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$$



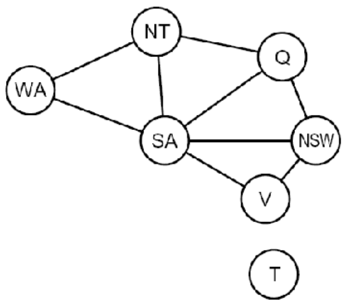
- Solutions are assignments satisfying all constraints, e.g.:

$$\{WA = red, NT = green, Q = red,$$
$$NSW = green, V = red, SA = blue, T = green\}$$

6

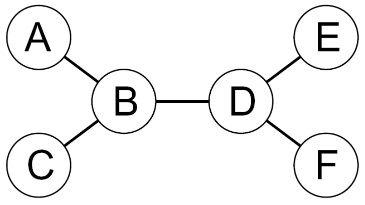
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



7

Tree-Structured CSPs

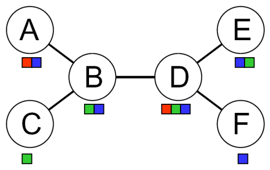


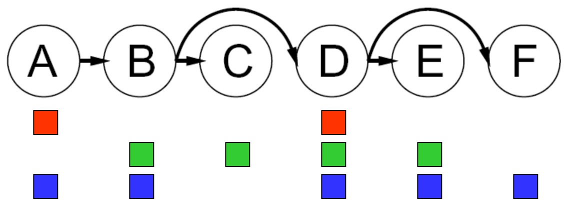
- **Theorem:** if the constraint graph has no loops, the CSP can be solved in O(n d^2) time
    - Compare to general CSPs, where worst-case time is O(d^n)
- This property also applies to probabilistic reasoning (later): an important example of the relation between syntactic restrictions and the complexity of reasoning.

8

Tree-Structured CSPs

- Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



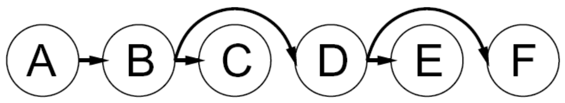


- For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)
- Runtime: O(n d^2) (why?)

9

Tree-Structured CSPs

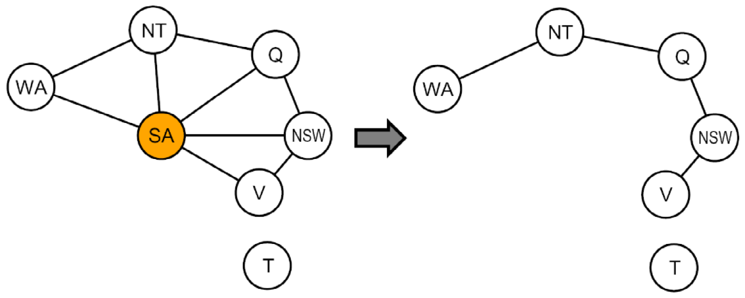
- Why does this work?
- Claim: After each node is processed leftward, all nodes to the right can be assigned in any way consistent with their parent.
- Proof: Induction on position



- Why doesn't this algorithm work with loops?
- Note: we'll see this basic idea again with Bayes' nets

10

Nearly Tree-Structured CSPs

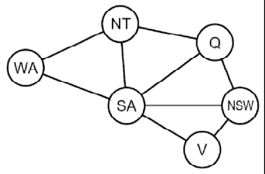


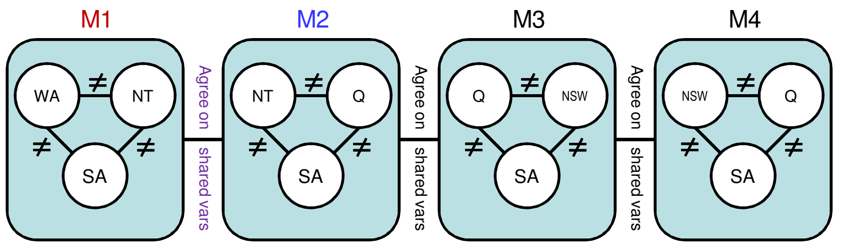
- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c) (n-c) d^2)$, very fast for small c

11

Tree Decompositions*

- Create a tree-structured graph of overlapping subproblems, each is a mega-variable
- Solve each subproblem to enforce local constraints
- Solve the CSP over subproblem mega-variables using our efficient tree-structured CSP algorithm





{(WA=r,SA=g,NT=b),
(WA=b,SA=r,NT=g),
…}

{(NT=r,SA=g,Q=b),
(NT=b,SA=g,Q=r),
…}

Agree: (M1,M2) ∈
{((WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)), …}

12

CSPs: our status

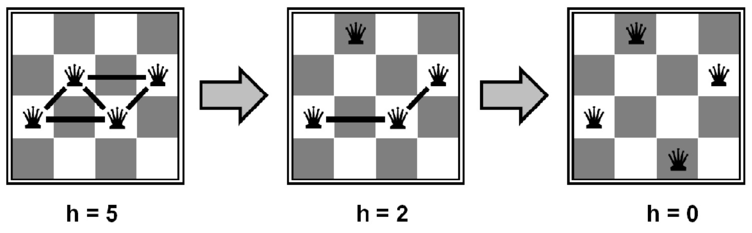
- So far:
  - CSPs are a special kind of search problem:
    - States defined by values of a fixed set of variables
    - Goal test defined by constraints on variable values
  - Backtracking = depth-first search with incremental constraint checks
  - Ordering: variable and value choice heuristics help significantly
  - Filtering: forward checking, arc consistency prevent assignments that guarantee later failure
- Today:
  - ✓ Structure: Disconnected and tree-structured CSPs are efficient
  - Iterative improvement: min-conflicts is usually effective in practice

13

Iterative Algorithms for CSPs

- Local search methods: typically work with “complete” states, i.e., all variables assigned
- To apply to CSPs:
  - Start with some assignment with unsatisfied constraints
  - Operators *reassign* variable values
  - No fringe! Live on the edge.
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - Choose value that violates the fewest constraints
  - I.e., hill climb with h(n) = total number of violated constraints

Example: 4-Queens

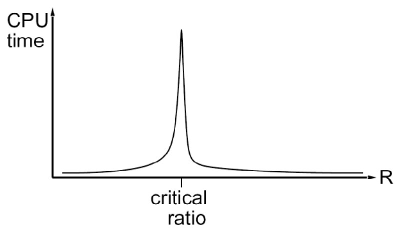


- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



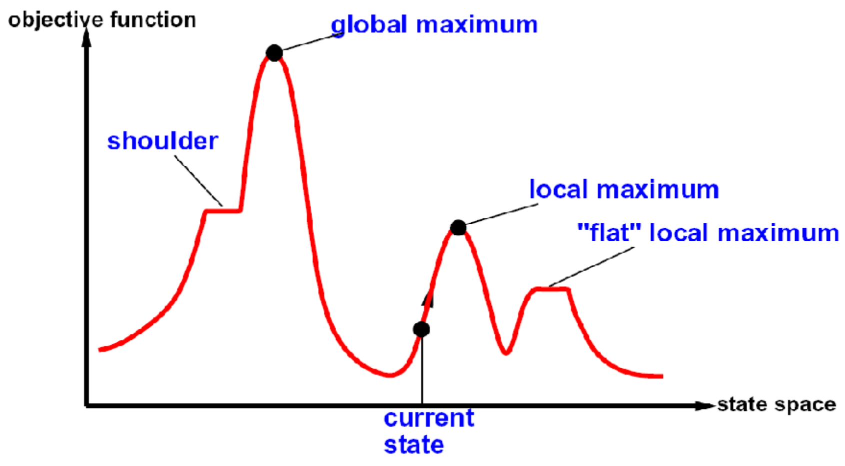
17

Hill Climbing

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What's good about it?

18

Hill Climbing Diagram



- Random restarts?
- Random sideways steps?

19

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

```
function SIMULATED-ANNEALING( $problem, schedule$ ) returns a solution state
  inputs: $problem$, a problem
          $schedule$, a mapping from time to "temperature"
  local variables: $current$, a node
                   $next$, a node
                   $T$, a "temperature" controlling prob. of downward steps

  $current \leftarrow$ MAKE-NODE(INITIAL-STATE[$problem$])
  for $t \leftarrow 1$ to $\infty$ do
      $T \leftarrow schedule[t]$
      if $T = 0$ then return $current$
      $next \leftarrow$ a randomly selected successor of $current$
      $\Delta E \leftarrow$ VALUE[$next$] – VALUE[$current$]
      if $\Delta E > 0$ then $current \leftarrow next$
      else $current \leftarrow next$ only with probability $e^{\Delta E/T}$
```

20

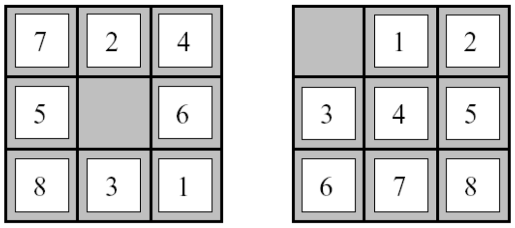
CSPs Summary

- CSPs are a special kind of search problem:
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking = depth-first search with incremental constraint checks
- Ordering: variable and value choice heuristics help significantly
- Filtering: forward checking, arc consistency prevent assignments that guarantee later failure
- Structure: Disconnected and tree-structured CSPs are efficient
- Iterative improvement: min-conflicts is usually effective in practice

21

22

Intermezzo: A* heuristics --- 8 puzzle



Start State Goal State

- What are the states?
- What are the actions?
- What is the cost?

Intermezzo: A* heuristics --- 8 puzzle

- Number of misplaced tiles: Admissible or not?
- What if we had an easier 8-puzzle where any tile could slide any direction at any time, ignoring other tiles and used their total *Manhattan* distance. Admissible or not?
- What if we had a piece of code that could quickly find a sequence of actions that reaches the goal state. Is the number of actions returned by that piece of code an admissible heuristic?

Intermezzo: A* heuristics --- pacman trying to eat all food pellets

- Consider an algorithm that takes the distance to the closest food pellet, say at (x,y). Then it adds the distance between (x,y) and the closest food pellet to (x,y), and continues this process until no pellets are left, each time calculating the distance from the last pellet. Is this heuristic admissible?
- What if we used the Manhattan distance rather than distance in the maze in the above procedure?

25

Intermezzo: A* heuristics

- A particular procedure to quickly find *a* perhaps suboptimal solution to the search problem is in general not admissible.
  - It is only admissible if it always finds the optimal solution (but then it is already solving the problem we care about, hence not that interesting as a heustic).
- A particular procedure to quickly find a perhaps suboptimal solution to a relaxed version of the search problem need not be admissible.
  - It will be admissible if it always finds the optimal solution to the relaxed problem.

26

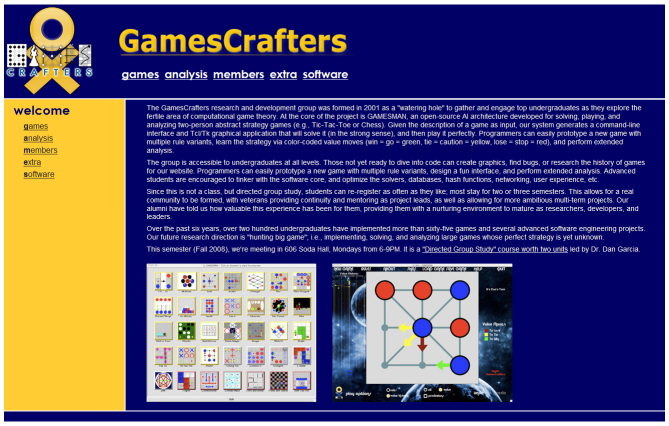
27

Game Playing State-of-the-Art

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions. Checkers is now solved!
- **Chess:** Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic.
- **Othello:** Human champions refuse to compete against computers, which are too good.
- **Go:** Human champions are beginning to be challenged by machines, though the best humans still beat the best machines. In go, b > 300, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.
- **Pacman:** unknown

28

GamesCrafters


GamesCrafters

games analysis members extra software

welcome

games
analysis
members
extra
software

The GamesCrafters research and development group was formed in 2001 as a "watering hole" to gather and engage top undergraduates as they explore the fertile area of computational game theory. At the core of the project is GAMESMAN, an open-source AI architecture developed for solving, playing, and analyzing two-person abstract strategy games (e.g., Tic-Tac-Toe or Chess). Given the description of a game as input, our system generates a command-line interface and Tcl/Tk graphical application that will solve it (in the strong sense), and then play it perfectly. Programmers can easily prototype a new game with multiple rule variants, learn the strategy via color-coded value moves (win = go = green, tie = caution = yellow, lose = stop = red), and perform extended analysis.

The group is accessible to undergraduates at all levels. Those not yet ready to dive into code can create graphics, find bugs, or research the history of games for our website. Programmers can easily prototype a new game with multiple rule variants, design a fun interface, and perform extended analysis. Advanced students are encouraged to tinker with the software core, and optimize the solvers, databases, hash functions, networking, user experience, etc.

Since this is not a class, but directed group study, students can re-register as often as they like; most stay for two or three semesters. This allows for a real community to be formed, with veterans providing continuity and mentoring as project leads, as well as allowing for more ambitious multi-term projects. Our alumni have told us how valuable this experience has been for them, providing them with a nurturing environment to mature as researchers, developers, and leaders.

Over the past six years, over two hundred undergraduates have implemented more than sixty-five games and several advanced software engineering projects. Our future research direction is "hunting big game"; i.e., implementing, solving, and analyzing large games whose perfect strategy is yet unknown.

This semester (Fall 2008), we're meeting in 606 Soda Hall, Mondays from 6-9PM. It is a "Directed Group Study" course worth two units led by Dr. Dan Garcia.

http://gamescrafters.berkeley.edu/

Run by Dan Garcia. Full for Spring. Check in with him for Fall 2010.

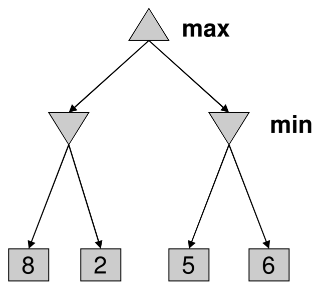
29

Game Playing

- Many different kinds of games!
- Axes:
  - Deterministic or stochastic?
  - One, two, or more players?
  - Perfect information (can you see the state)?
- Want algorithms for calculating a strategy (policy) which recommends a move in each state

31

Simple two-player game example



32